Modeling VLSI Nets as RC Networks

Augmenting Circuit Representations With Geometric Information

Benjamin Goldstein

Advisor: Professor Rajit Manohar

Submitted to the faculty of the Department of Computer Science in partial fulfillment of the requirements for the degree of Bachelor of Science

Department of Computer Science
Yale University

December 3, 2023
[Revised December 14, 2023]

Abstract

In the age of artificial intelligence, highly specialized integrated circuit design is ever important. These devices are of truly awesome scale: Apple's latest M1 Ultra recently set a high watermark with over 110 billion transistors [1]. At this scale most precise circuit analysis techniques are simply untenable from a compute perspective. Electromagnetic field solvers commonly used for smaller circuits are simply not up to the task—new heuristics are needed. This paper proposes a method of storing a circuit design's geometric structure, and leveraging such information to generate a resistor/capacitor (RC) network.

In VLSI, entities called "nets" describe the connectivity of a circuit. These nets are effectively a series of wires with determined routes, with each net connecting to some set of pins. A simple heuristic for net analysis would be to consider these nets as ideal short circuits, as this is typically a reasonable approximation for a metal wire. This neglects two key parasitic effects: resistive effects from non-ideal conductivity, and capacitive effects with either the substrate or nearby wires. RC networks have simple solutions to calculate wire delay, a key metric in circuit analysis, and are therefore good approximations of behavior for many applications. The method proposed processes net information to generate a sequence of rectangles corresponding to the true geometry of the wire. The rectangles are then stored in a geometric data structure and used to create an RC network for downstream analysis.

The implementation is integrated with an existing toolset developed by the Yale Asynchronous VLSI (Very Large Scale Integration) and Architecture group, led by Professor Manohar. The physical database, or `PhyDB`, repository contains a set of utilities that store a custom representation of a design given industry standard design files [2]. While the pipeline is not yet a complete end to end solution it provides a solid foothold for future work in bringing geometric RC analysis into `PhyDB` and related tools.

Table of contents

Front matter
Abstract i
Table of contents ii
Acknowledgements iii

1 Introduction **1**
1.1 VLSI Background 1
1.2 PhyDB Background 3
1.3 Project Introduction 4

2 Methods **6**
2.1 Geometric Augmentation of Net Handlers 6
2.1.1 LEF/DEF Parser Configurations + Assumptions 6
2.1.2 Centerline Partition Scheme 8
2.1.3 Geometric Data Structure: Binning 9
2.2 RC Network Generation 10
2.2.1 Distributed RC Circuit Modeling 11
2.2.2 Resistance Network Generation 12
2.2.3 Coupling Capacitance Generation 14

3 Discussion & Conclusion **16**
3.1 Results/Deliverables 16
3.2 Further Work 16
3.3 Conclusion 18

Acknowledgments

There are many people I'd like to thank for the success of this project. First and foremost, my advisor Professor Rajit Manohar provided mentorship that was integral in making this project happen. The constant access to his unique domain expertise kept me unblocked and well equipped to tackle difficult technical problems.

I'd also like to Professor Sohee Park and the rest of the course staff of CPSC 490. They provided structure and modularity to a task that otherwise would have proved overwhelming. The lecture series was informative, and hearing updates regarding the projects of others was both interesting and instructive to my own work.

Finally, I'd like to thank my family and friends who've stood by me along the way! Without their unflinching support this simply would not have been possible.

Chapter 1

Introduction

Before detailing the particular methodology some requisite materials must be introduced. This chapter will attempt to do just that, beginning with an overview of relevant Very Large Scale Integration (VLSI) terminology and concepts, moving into the existing pertinent toolset developed by the Yale AVLSI group, and finally setting up the project itself with proper context. Basic familiarity with electrical concepts (i.e. voltage, resistance, and capacitance) is assumed, but particular details about circuit design and representation are included.

1.1 VLSI Background

The principal purpose of a VLSI design is to create a representation of a very large scale integrated circuit that can then be fabricated to realize certain functionality. The fabrication process, vastly oversimplified, consists of etching rectangles of different materials in layered fashion. Therefore the result of the VLSI design specifies the locations, dimensions, layers, and materials of a set of rectangles. In aggregate, this rectangle set suffices to describe the circuit in its entirety. These circuits, generally speaking, consist of elements that are equipped with pins for inter-element and off-board connections. An example of an element could be a cell that implements a logical AND gate. Such a cell would likely need pins that

correspond to inputs, output, power, and ground. Pins connected by a wire are said to share a signal, as at least ideally the wire instantaneously maintains voltage equality. Continuing along with the AND gate example, the pins of the AND gate would have to be connected to appropriate locations around the circuit with wires. At the scale of these designs, with billions of transistors, the wiring patterns are incredibly intricate. Pins are connected with wiring patterns grouped into electrically connected networks called nets. It is these nets that are the focus of this analysis, as wires have nonzero resistivities (lending themselves to an Ohmic resistor model), and nearby wire pairs exhibit coupling effects that can be modeled as a parallel-plate-like (or otherwise constant capacitance) connection.

VLSI designs are commonly stored in a pair of files: Library Exchange Format (LEF) and Design Exchange Format (DEF). The former is focused on an abstract view of the design: i.e. what the components are, rules about the underlying technology being used, and their connectivity (but not routing information). That is, LEF files give an operating picture about what the circuits components are, how they're connected, and and some higher level ground rules, but lack detailed information about precise geometric structure. Since the focus of this project is on geometric analysis of routed nets (nets whose precise paths are specified), the LEF file mainly serves as a reference for definitions and other values regarding material properties. The DEF file, on the other hand, contains low level information about the exact routes of the circuit. Given the importance of the nets' geometry to this analysis, this file (particularly the net section) is the primary area used in this project.

The routing of a net is given by a series of coordinates that specify the centerline path of the wire. There is support for both 90 degree and 45 degree turns in the DEF specification, but this project only supports 90 degree, or "orthogonal" routing patterns. The wire zig zags about this path with a specified width, either explicitly specified or taken from the layer's default, and extends beyond its endpoints with a specified extension length, again either explicitly specified or taken to be half the wire's width. A net can also contain rectangles of arbitrary dimension, as well as interconnects between layers. These interconnects, called vias,

are generally just rectangles drawn on three layers: the bottom metal layer, an intermediate "cut" layer, and the top metal layer. There are more complex via configurations, but those are outside the scope of this project. With this information in mind it should be clear that the DEF contains sufficient information to generate a set of rectangles associated with a given net.

1.2 PhyDB Background

The Yale AVLSI group focuses on creating novel VLSI designs using asynchronous architectures. Most modern circuits run in synchronous fashion using a "clock" signal that functions as the pulse of the device. There are various benefits to getting rid of this signal allowing asynchronous operation, including speed and energy efficiency. Commercial tools for VLSI are closed source and deeply entangled in an anticompetitive tangle of intellectual property disputes. Such an anticompetitive structure has stifled innovation in the area of design tools, and the state of the art tools that do exist require expensive licenses to use. Given this suboptimal state of affairs, the AVLSI group has developed a suite of open source tools that it uses in its own design process. These tools provide a transparent, publicly available alternative to VLSI designers of all sorts, both in and out of the group. Thus, an augmentation of this codebase provides not only the group but also the public with new functionality that can impact future technological progress.

Specifically, the AVLSI tool being extended in this project is `PhyDB`. This is the physical database representation of the VLSI circuit: a data structure containing the information contained in the LEF/DEF files. For efficiency and compatibility with necessary dependencies, `PhyDB` is implemented in C++. `PhyDB` loads the information from the LEF/DEF files using an industry standard open source parser [4]. The parser follows a callback registration design: when it encounters any of the entities defined in its syntax manual (including nets, vias, and many many other constructs not mentioned in this brief introduction) it loads

corresponding properties into an associated C++ object and executes a callback to process it. PhyDB, therefore, implements a series of methods that handle occurrences of relevant constructs in the LEF/DEF files.

The principal callbacks of concern are registered by `defrSetNetCbk`, and `defrSetSNetCbk`. The former handles occurrences of standard nets, while the latter handles occurrences of "special nets": nets that connect pins with special status like power and ground. Prior to this project, the net handler simply registered the net's pin names into the database. The specialnet handler adds path information to the database object (this is done by default for standard nets) but does not process it further. To create an RC model of the circuit's nets it is necessary to ingest geometric information from both nets and special nets at this step and store a set of rectangular "wire segments" that correspond to the true geometry of the design.

1.3 Project Introduction

Now that some terminology and existing tooling have been clarified the goals of the project can be lain out in more precise terms. First and foremost, the net and special net handlers must be modified to ingest geometric information and store it in `PhyDB`. This will require traversing the paths of each wire of each net (this is the hierarchy of the parser's API). This traversal will contain sufficient information to partition the net into a series of rectangles, which can then be stored in a geometric data structure inside of `PhyDB`. From there, `PhyDB` can be extended to support RC network generation. This step will convert the rectangle set into a non-geometric RC network consisting. The nodes are labeled by their net and an additional identifier, and connected by capacitances or resistances. A complete RC analysis would be more expensive than its worth: the computational cost of analyzing the capacitance between each pair of wire segments scales with n^2 where n is the number of segments in the circuit. This is untenable at the scale of these designs, so some heuristics must be used. The

data structure to be implemented will use a partitioned approach, making it simple to query the neighborhood of a given segment. Excluding distant segments with negligible coupling effects grants massive computational reprieve at minimal accuracy cost. It should also be noted that precise capacitance calculations would require the usage of an electromagnetic field solver. To use a solver on each pair would also make this analysis computationally intractable. A proposed approach would be to precompute a set of template capacitors using a solver, then interpolate capacitance estimates of segment pairs in the circuit given these templates. This falls outside the scope of the project, though, and could represent a fruitful area of further research.

Chapter 2

Methods

This chapter concerns the design and implementation details of the RC analysis tool as described in Section 1.3. The implementation is divided into two principal components: the augmentation of net-handling callbacks to store rectangular wire segments in a geometric data structure, and then RC network generation.

2.1 Geometric Augmentation of Net Handlers

The hierarchical structure provided by the LEF/DEF parser was briefly mentioned in the introduction but demands further elaboration here. Both the regular and special net handlers are passed a pointer to a `defiNet` object newly created by the parser. The callbacks are also passed a third pointer, called `defiUserData`, which is an attribute of the parser configuration. `PhyDB` has set this pointer to a reference to the current database object, so the callbacks all have access to the database they are designed to populate.

2.1.1 LEF/DEF Parser Configurations + Assumptions

Each `defiNet` has a series of `defiWire` objects that are each a thin wrapper on a series of `defiPath` objects. Each `defiPath` has an interface for traversal, with each element of

the path being accessible via an API that depends on its type. For instance, if the path element is of type layer, the name of the layer is accessible via `defiPath::getLayer`. The functionality being added focuses on the following path element types: `DEFIPATH_LAYER`, `DEFIPATH_VIA`, `DEFIPATH_WIDTH`, `DEFIPATH_FLUSHPOINT`, and `DEFIPATH_VIARECT`. The assumptions surrounding the handling of these paths are outlined below:

1. The layer is set before anything else, as it defines which layer subsequent features occur on.
2. A sequence of points represents a centerline of a portion of a wire. Its width is either the default for its layer or determined by the most recently seen `DEFIPATH_WIDTH` element since the last flush.
3. The centerline is "flushed", i.e. processed & inserted into the data structure and subsequently reset, when either
   (a) A layer change is detected, i.e. a `DEFIPATH_LAYER` is detected and a non-null layer is already set.
   (b) A path ends.

   The width and extension length are reset after each flush, though the layer only is reset after each path end.
4. The extension length is either the default for the net (half wire width for net, 0 for special net) or determined by the most recent third coordinate of a `DEFIPATH_FLUSHPOINT` seen since the start of the sequence. Both endpoints of the wire sequence share an extension length.
5. `DEFIPATH_VIA` and `DEFIPATH_VIARECT` occur only after a single point that determines the position of these elements. Multi-point sequences before these elements do not occur.
6. Path element types not listed either (a) do not occur or (b) do not affect the geometry of the net's wiring.

Some of these assumptions follow from design rules/the LEF/DEF specification, and others are permissibly violated by a valid design. The handler is thus a proof of concept that is only guaranteed to be correct for designs that satisfy these assumptions. The tool functions correctly on a subset of designs, including many practical large scale ones, but requires further work to make fully correct for all permissible cases under the LEF/DEF specification.

2.1.2 Centerline Partition Scheme

When a via or rect geometry is added to `PhyDB` it is geometrically simple: each of these cases leads to a fully specified rectangle or set of rectangles that can be directly added. When a centerline is flushed, though, the rectangle or series must be generated via some partitioning scheme. Below an example partitioning scheme is shown by considering a series of points $(x_1, y_1), (x_1, y_2), (x_2, y_2)$ that constitute a wire portion with width w and extension length e:

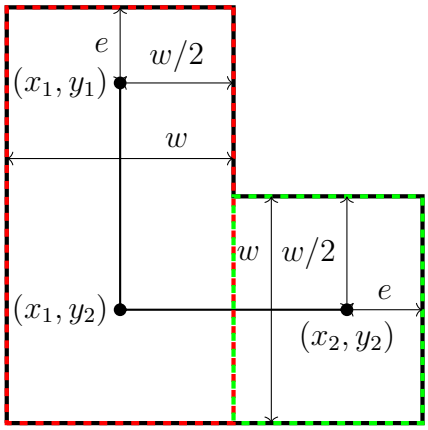

Figure 2.1: Illustration of partition behavior for three point centerline

Note that this scheme is fully generalizable to any orthogonally routed sequence: the left and right ends are set up like (x_1, y_1) and (x_2, y_2) respectively, and junctions are handled like (x_1, y_2). The full L-shaped region outlined in black is partitioned into two rectangles, R_1 outlined in red dashes and R_2 outlined in green. In order to make the rectangles non-overlapping, the first rectangle greedily consumes the area past its endpoint. The second rectangle's starting edge is therefore between its start point and endpoint. This scheme ap-

plies to longer point sequences, as the start edge each rectangle yields area to its predecessor while the end edge extends beyond the endpoint. It should also be clear that the extension length for the endpoints is different from that of the internal points in the sequence as desired, though if $e = w/2$ as is the default for standard nets then this distinction is lost.

2.1.3 Geometric Data Structure: Binning

The rectangles, either from a partitioned centerline, via definition, or most directly from a rect element, must then be organized. A new C++ class is created to represent these rectangles called the `WireSegment`. A `WireSegment` contains a net name, layer name, rectangle, segment number, and references to connected segments. Each wire segment stores a reference to (a) segments that precede it in a centerline partition and (b) segments that are below it in a via. These stored connections are used to facilitate the creation of resistors without reprocessing the full net to discern connectivity. The segment number is a unique identifier for a segment when combined with the net name. Each segment therefore is uniquely identifiable and contains sufficient information to track both intra-layer and inter-layer connectivity within the net.

The proposed data structure utilizes binning to partition the rectangles into spatial regions per-layer. That is, for each layer a grid is set up such that each square of the grid contains a list of pointers to any wire segment with a rectangle that intersects the grid square. This facilitates neighborhood queries of a given segment: i.e. obtaining any other wire segment registered to a bin that either is or borders a bin the segment is registered to. Since the space will be somewhat sparse a map between pairs of `partition_id`'s is used rather than a matrix. The pair contains one `int` corresponding to the `x` coordinate and the other for the `y`, each determined by dividing the coordinate by bin width. An example of this structure is shown below:

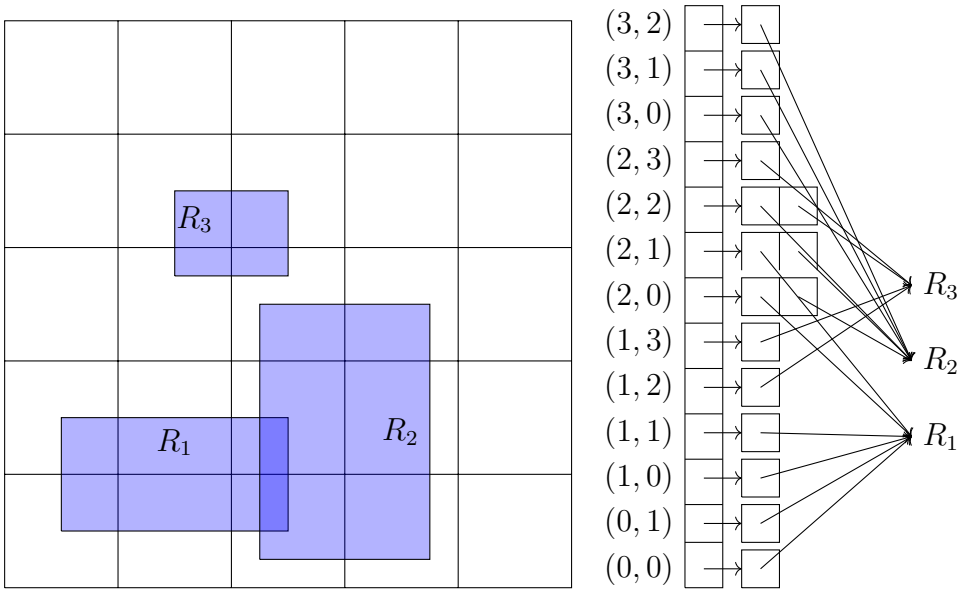

Figure 2.2: Visual (left) and memory (right) representations of rectangle binning example

A given wire segment will be registered to all bins with `x_id` and `y_id` between those of its lower left corner and its upper right inclusive of the endpoints, as shown above. Also, each rectangle (or in our case wire segment) is heap allocated only once, and C++ smart pointers are used to avoid excessive copying as well as manual memory management. A final note is that the geometric data structure will also maintain a map of net name to a set of wire segments, as the RC networks will be generated by iterating over each net. Resistances will derive from intra-net connections, while capacitances will follow from inter-net coupling effects. The RC generation component of the project now has sufficient support from the PhyDB data structure.

2.2 RC Network Generation

With a geometric data structure in place, it is now possible to generate an RC network. The proposed workflow is to begin by generating a resistor network for each net, and then inserting capacitances between sufficiently coupled segment pairs across nets. Capacitances to the substrate were excluded from this project, but are an are for further exploration.

Resistance values are estimable using the material and dimensions of the segment, but precise calculation remains out of the scope of the project. Ahead of delving into precise methods, an overview of how exactly capacitively coupled resistive wires are modeled is certainly warranted.

2.2.1 Distributed RC Circuit Modeling

The problem at hand is how to model a distributed RC circuit: a pair of segments that have resistive and capacitive properties that are distributed across the whole of their lengths. Diagrams of the configuration along with three proposed models is shown below:

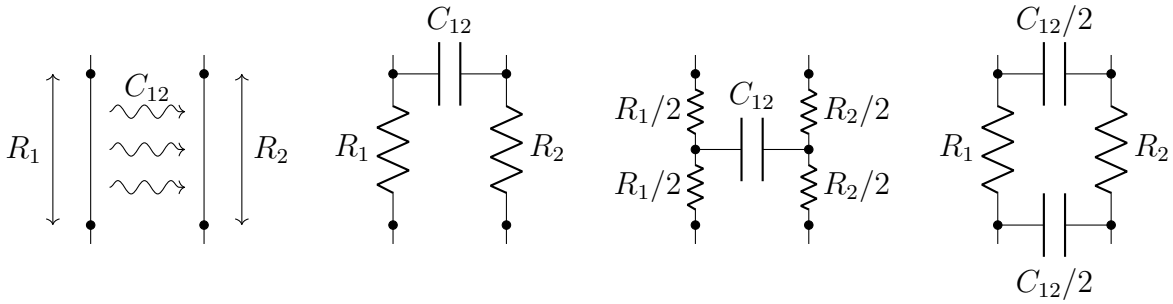

Figure 2.3: Left to right: circuit configuration, L model, T model, and π model

It turns out that the L model is particularly poor at estimating wire delay, one of the key motivations for the RC analysis in the first place, so it is disregarded from here on out [3]. The T model divides the resistances in half while introducing a central capacitance. The π model, conversely, divides the capacitances in half introducing one at each end of the resistor pair. Either of these is fine for delay estimation, and both are conducive to further recursive splits for increased accuracy. Recursive splits of each viable model are shown below:

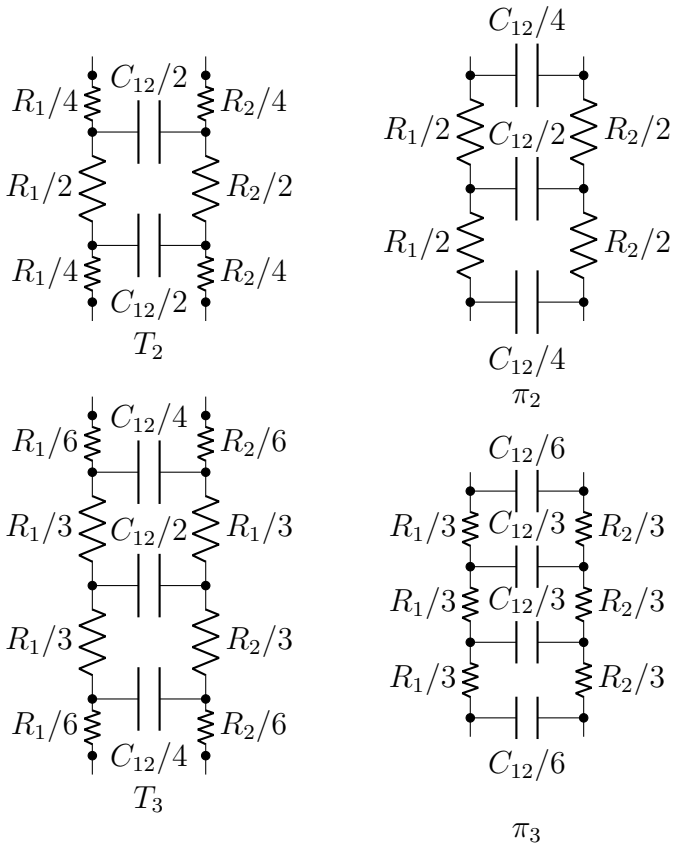

Figure 2.4: Second and third recursive splits of T (left) and π (right) models

Given that the π model introduces an additional capacitor, the T model is used for ease in this project. Given the procedure outlined for RC network generation it is better to minimize the number of capacitances introduced as these require re-splitting resistors.

2.2.2 Resistance Network Generation

The task at hand is now to generate a resistor network from the rectangular segments stored as described in Section 2.1.3. Each rectangle is modeled as a point to point resistance between its start and end coordinates, which are specified in the wire segment. The start coordinate of a wire is equal to the center of its starting edge (unless its the first segment in the sequence, then start coordinate is equal to centerline coordinate), and the end coordinate is equal to centerline coordinate. When a rectangle has a neighboring rectangle, an "escape

resistance" is used to model the connection as shown in the figure below which recycles the example configuration from 2.1:

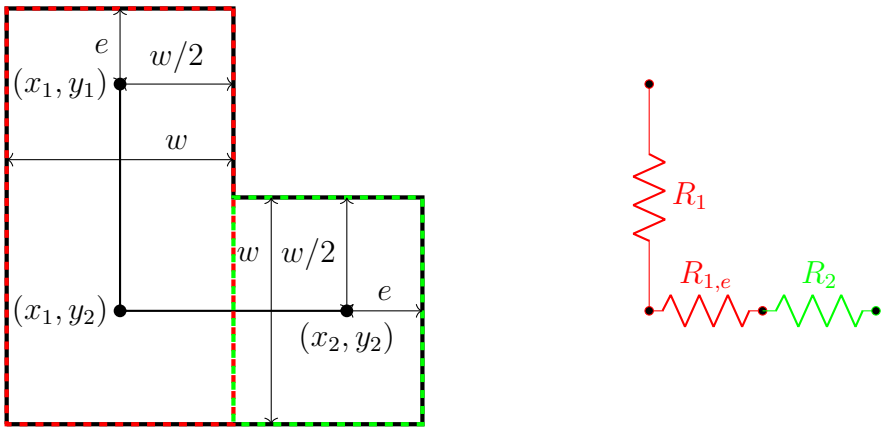

Figure 2.5: Illustration of conversion from partition to resistance net

Where $R_{1,e}$ is the "escape resistance" between R_1's lower centerline coordinate and the center of R_2's starting (left) edge. This particular model of resistance is used to grant a unique `WireSegment` owner to each created resistor, which aids downstream in the capacitance process. Despite the irregular structure of the corner escape resistances are modeled as having wire width w for consistency. It should be noted that this assumes connectivity to follow from the centerline coordinates; not just any two neighboring metal rectangles could follow this model. This methodology does not include vias, which are tackled separately. This resistance generation uses the connectivity information stored during centerline partitioning to generate escape and principal resistances for each segment. Segments with via connectivity are also connected by resistances computed from the via dimensions and information taken from the cut layer. Another significant point is that although the final network does not retain geometric character, it is necessary for the resistance network to maintain a notion of geometry to facilitate capacitance introduction. Therefore, the coordinates used to compute the resistances are also tracked in the data structure. These resistances are defined by the material, length, and width of the corresponding metal segment they represent.

Via and rect resistances fall into a different category, which will be called `VerticalResistor`.

These special resistors do not correspond to current flow in the plane, but rather to inter-layer "vertical" current. These resistors are therefore defined by their cross sectional area, i.e. the rectangular area of the segment. They also lack point-to-point representation. Once explicit vertical and horizontal connections have been represented with standard and vertical resistors, pairwise operations are conducted on the net to detect non-standard overlaps. Rectangles that are fully contained in others are coalesced, while overlapping pairs are connected with additional resistances. The overlapping configuration is common when a via is connected to a standard wire segment, as these will occur on separate paths and thus not be explicitly connected at parse-time. The end result of this stage is a connected resistor network for each net.

2.2.3 Coupling Capacitance Generation

With a resistance network in place it is now possible to introduce capacitances. The process begins with an iteration over each wire segment in the design. A neighborhood query is conducted on each segment, excluding segments from the same net. For each nearby segment of another net, the overlap area and distance are calculated using the segments' rectangles. This is the point where a capacitance could be interpolated from a template, but for now these dimensions are just logged in the data structure. The overlap interval between the two segments will lie within some set of resistors, likely one, per segment. If there are multiple resistors then the capacitance must be split up into subcapacitances, such that each subcapacitance corresponds to one resistor pair. Then, the T model is applied to model the capacitance in the circuit. The result is a newly partitioned resistor set associated with each segment involved in the coupling. Each capacitance is then simply a pair of resistor nodes (along with the stored overlap area and distance dimensions) that can be stashed in a set to later be reported. For simplicity the T model is only applied to one iteration, but it is clear that recursive application is straightforward to add in future work with the depth being a configurable parameter of the database.

The way the T model is applied is as follows. If the center of the full overlapping region between two segments falls within a resistor, that is the point at which it is added to the network. If this is not the case, which could be due to the fact that the center of the overlapping region is near the corner of an L-shaped junction, then an alternative approach is used. If the overlapping region itself overlaps with any resistor, then the center of that region is used as the attachment point. If that is not the case then the capacitor is considered insufficiently overlapping to be included in the model, as the region of interest doesn't coincide with any modeled resistors. An alternative heuristic, though, would be to attach the capacitor at the nearest resistor node.

Once all of the coupling capacitances have been added, the result is an RC network model for the circuit design. It is also feasible to generate capacitances from each segment to the substrate, though without concrete values all this introduces is clutter. A simple reporting mechanism is included that dumps all resistors and capacitors to output, with a pair of node identifiers along with the resistance in the resistor case or relevant dimensions in the capacitor case to output. An example of a resistor, vertical resistor, and capacitor output are included below:

```
Resistor<node1='var_x.w.wt[12]{2}', node2='var_x.w.wt[12]{26}', length=1800,
        width=300, layer=li, segment-id='var_x.w.wt[12]:1'>
VerticalResistor<lower-node='GND{772}', upper-node='GND{775}',
        cross-sectional-area=202500, layer=v3, segment-id='GND:3'>
Capacitor<node1='e2._passthru_.new_n131_{45}', node2='L.d[25]{146}',
        overlap-length=225, distance=1050>
```

It is now clear how RC nodes and segments are identified. Note it is clear why capacitors don't have a segment identifier as they span multiple segments. This output forms the backend of the pipeline, and represents the culmination of the project!

Chapter 3

Discussion & Conclusion

This chapter serves to summarize the results of the project and further work that remains toward a complete solution to this analysis problem.

3.1 Results/Deliverables

The final product of the project is an end to end pipeline that extends `PhyDB` to enable RC network reporting. The pipeline is able to run end to end on example LEF/DEF files, though at times runtime scalability is a concern. The source code for the tool is available at this fork of the PhyDB github repository, which though not ready for merge into the open source toolset, shows the complete code contribution aspect of the project. It is notably difficult to test the correctness of the pipeline, as the standard of comparison is the plaintext spec. As a result a comprehensive test suite is not part of this project and remains an area of potential future research.

3.2 Further Work

There have been sufficient mentions regarding further work to warrant its own section in this report. First, a well designed approach to utilizing precomputed results from a field

solver would likely serve as the best form of capacitance estimation, and would bring the RC network from a conceptually interesting to practically useful entity for circuit analysis. Resistances should also be calculated using information about layer thickness and material resistivities. The RC model should likely also be revisited with some validation procedure in mind to ensure that it accurately estimates the quantities it was designed to. Whether or not this particular structure of rectangle-based resistance modeling preserves relevant circuit behavior is not at all guaranteed. Furthermore, a significant amount of validation must go into the rectangle generation portion of the pipeline to ensure that it is (a) correct on the subset of LEF/DEF inputs that it seems to be and (b) well covers a sufficiently wide subset of possible inputs to be widely usable.

At that point functionality should also be added to warn or fail gracefully in particular circumstances that may violate assumptions. Additionally, mapping the RC network (now arbitrarily labeled) onto the signals and pins of the design will be necessary for any practical analysis to occur. This is feasible given the information in the LEF/DEF inputs but may require complex queries of existing `PhyDB` data and/or additional callback functionality to better store the needed information at parse-time. This model also only includes planar coupling capacitances, and a potential improvement would include inter-layer coupling as well as capacitance to the substrate for each segment. Finally, with a correct RC model properly mapped onto labeled signals of the design, an export operation to SPICE or another simulation format should be implemented to truly bring the end to end to fruition. There is also absolutely room for further optimization of the data structure—performance with quadtrees, kd-trees, or other geometric structures should absolutely be examined. This project provides a foundation that can be built upon to render this tool fully functional and widely deployable.

3.3 Conclusion

This project introduces geometric information into an open source database representation of physical circuit designs. It includes novel procedures for wire partitioning and resistance network generation from a set of wire segments. The pipeline culminates in a report of a global resistance network, providing a valuable proof of concept and stepping stone toward a usable RC network model of a VLSI netlist. This work contributes to the enablement of rapid, accurate, performance modeling of large scale electronics, a prospect that will accelerate the design of the next generation of specialized integrated circuits.

Bibliography

[1] Apple. 2022. Apple unveils m1 ultra, the world's most powerful chip for a personal computer. *apple.com* URL https://www.apple.com/newsroom/2022/03/apple-unveils-m1-ultra-the-worlds-most-powerful-chip-for-a-personal-computer/.

[2] AVLSI, Yale and Architecture Group. 2023. Phydb. https://github.com/asyncvlsi/phydb.

[3] Horowitz, Mark. 2007. Lecture 2: Wires and wire models. URL https://web.stanford.edu/class/archive/ee/ee371/ee371.1066/lectures/lect_02.2up.pdf.

[4] Systems, Cadence Design. 2008. Lefdef. https://github.com/asyncvlsi/lefdef.